




# MongoDB - Crear Cuenta



Aceptamos los términos de licencia y contestamos a las preguntas que nos hacen sobre el uso que pretendemos hacer.

 MongoDB.

Accept Privacy Policy & Terms of Service

Please acknowledge the following terms and conditions to finish creating your account.

☒ I accept the [Privacy Policy](#) and the [Terms of Service](#)

Cancel Signup

Submit



# MongoDB - Seleccionar servidor



Seleccionamos el plan gratuito, el proveedor que más os guste, la región que os pille más cerca y le ponéis un nombre al servidor.

**M10****\$0.08/hour**

For production applications with sophisticated workload requirements.

STORAGE

10 GB

RAM

2 GB

vCPU

2 vCPUs

**SERVERLESS****\$0.10/1M reads**

For application development and testing, or workloads with variable traffic.

STORAGE

Up to 1 TB

RAM

Auto-scale

vCPU

Auto-scale

**M0****FREE**

For learning and exploring MongoDB in a cloud environment.

STORAGE

512 MB

RAM

Shared

vCPU

Shared

Provider

Region

★ Recommended region ⓘ

N. Virginia (us-east-1) ★

Name

You cannot change the name once the cluster is created.



# MongoDB - Crear usuario e IPs



En la opción usuario y contraseña creáis un nombre de usuario y la contraseña que es MUY IMPORTANTE que apuntéis, en el paso 2 seleccionamos entorno local y las IPs que permitís que se conecten. 0.0.0.0 serían todas las IPs

### Security Quickstart

To access data stored in Atlas, you'll need to create users and set up network security controls. [Learn more about security setup](#)

1 How would you like to authenticate your connection?

Your first user will have permission to read and write any data in your project.

Username and Password

Certificate

**i** We autogenerated a username and password for your first database user in this project using your MongoDB Cloud registration information. **x**

Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

**Username**

**Password**

 [Autogenerate Secure Password](#) [Copy](#)

**Create User**

2 Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

**My Local Environment**  
Use this to add network IP addresses to the IP Access List. This can be modified at any time.

**Cloud Environment** ADVANCED  
Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

**Add entries to your IP Access List**

Only an IP address you add to your Access List will be able to connect to your project's clusters.

IP Address	Description
<input type="text" value="Enter IP Address"/>	<input type="text" value="Enter description"/>
<input type="button" value="Add My Current IP Address"/>	
<input type="button" value="Add Entry"/>	



# MongoDB - Acceso al servidor



Ésta es la parte que necesitáis para conectar node con la base de datos.

## 3. Add your connection string into your application code

☒ View full code sample

```
mongodb+srv://dorian-trazos:<password>@mongodb-trazos.ca3bdxj.mongodb.net/?  
retryWrites=true&w=majority
```





# MongoDB - mongoose



---

Para interactuar con MongoDB de una forma cómoda existe el módulo de mongoose, lo instalaremos con `npm install mongoose`, y después en nuestro servidor lo usaremos en el archivo principal.

```
const mongoose = require('mongoose')
```



# Mongoose - Conectar servidor



Es recomendable conectar la base de datos antes de levantar el servidor, por lo que haremos una función asíncrona que conecte la base de datos antes de poner al servidor en escucha.

```
const startServer = async () => {  
  try {  
    await mongoose.connect(process.env.MONGODB_URL);  
    console.log('Connected to database');  
  } catch (err) {  
    console.error(`Connection error`);  
  }  
  
  app.listen(  
    process.env.PORT,  
    console.log(`Server listen on port ${process.env.PORT}`)  
  );  
};  
  
startServer();
```



# Mongoose - Schema and model



Para trabajar con mongoDB hay dos conceptos principales a conocer.

Schema: Los esquemas son la estructura que tendrán nuestras entidades (el objeto a guardar)

Model: El modelo es la compilación de ese esquema para poder trabajar con la base de datos.

```
const mongoose = require('mongoose');

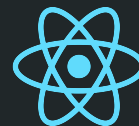
const UserSchema = mongoose.Schema({
  _id: String,
  name: String,
  email: String
});
```

```
const UserModel = mongoose.model('User', UserSchema);

module.exports = UserModel;
```

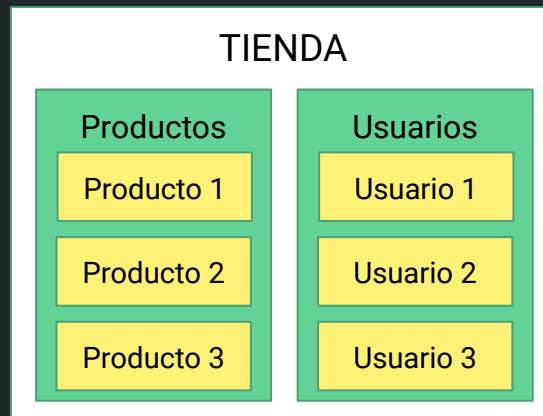
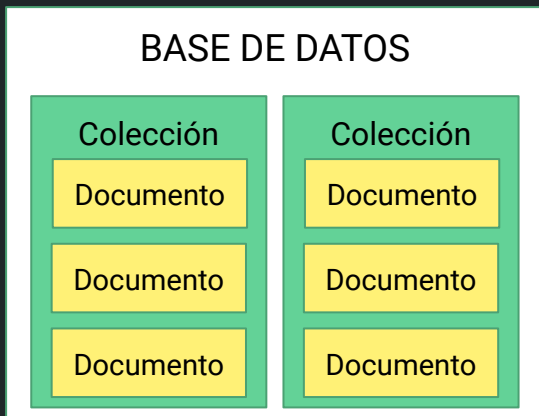


# MongoDB - Terminología



Colección: Son cada uno de los registros que tenemos en la base de datos.

Documento: Es cada uno de los registros de las colecciones.







# Mongoose - Leer documento



**TODAS** las operaciones con base de datos son **ASÍNCRONAS**, por lo que deben ir en una función **async**.

Para leer la base de datos tenemos la función **find()**, si no pasamos parámetros de búsqueda se interpreta como búsqueda sin filtros, por lo que nos devolverá todos los elementos de la base de datos.

```
// Obtener todos los usuarios
controller.allUsers = async (req, res) => {

  try {

    const allUsers = await UserModel.find();

    res.status(200).send(allUsers);

  } catch (err) {

    res.status(500).send({ error: 'Error al leer la base de datos' });

  }

};
```



# Mongoose - Crear documento



Para utilizar el modelo crearemos una instancia del modelo, para ello utilizamos el operador **new** junto con el nombre de nuestro modelo y pasaremos el objeto con los datos como parámetro.

```
app.post('/', async (req,res)=>{
  const user = await UserModel.findById(req.params.id)

  if(user){
    return res.status(409).send('User exist')
  }

  const {name, email} = req.body;

  const newUser = new UserModel({
    _id:v4(),
    name,
    email
  })

  await newUser.save()

  res.send('User registered')
})
```



# Mongoose - Editar documento



Para editar tenemos la función `updateOne({filter}, { $set: { newData } })`, el primer parámetro será el filtro del documento y el segundo los nuevos datos a actualizar.

```
app.patch('/', async (req,res)=>{
  const user = await UserModel.findById(req.params.id)

  if(!user){
    return res.status(409).send('User not exist')
  }

  await UserModel.updateOne({ _id: user.id }, { $set: { age: 26 } })

  await newUser.save()

  res.send('User updated')
})
```



# Mongoose - Borrar documento



Para borrar un elemento de la base de datos podemos usar la función `deleteOne({ filtro })`

```
app.delete('/', async (req, res) => {  
  const user = await UserModel.findById(req.params.id);  
  
  if (!user) {  
    return res.status(409).send('User not exist');  
  }  
  
  // Opción 1  
  await UserModel.deleteOne({ _id: user.id });  
  
  // Opción 2  
  await user.remove();  
  
  res.send('User deleted');  
});
```